2.0 DII COE Components

The concept of a Common Operating Environment (COE) is perhaps the most significant and useful technical by-product of the GCCS development effort. It represents the culmination of several years of development amongst the services/agencies, and it is interesting to note that the services/agencies independently arrived at similar conclusions. The DII COE encompasses architecture, standards, reusable software modules, and automated integration in a cohesive framework for systems development.

Yet the COE is perhaps the most misunderstood concept in the entire GCCS architecture. The COE is in reality very simple and straightforward, but powerful in its ability to tailor a system to meet individual site and operator requirements. Much of the confusion can be traced to uncertainty over what software is or is not in the COE, whether or not the COE and GCCS are synonymous terms (they are *not*), and the evolutionary development nature of the COE. The requirement to establish a migration strategy which preserves legacy systems and the near-term WWMCCS replacement objective are responsible for a great deal of confusion, but the need to evolve the COE is driven by many factors and the architectural freedom to evolve is one of the strengths of the COE.

This chapter is devoted to explaining the DII COE in detail. Definition of a COE is principles driven, not application driven, so this chapter begins with a discussion of those principles. Selection of the actual components which constitute a COE instantiation determine the specific problem domain that a COE can address (e.g., C4I for GCCS, logistics for GCSS), and how broadly defined the problem domain can be. Because the COE is structured so that only required components are loaded, a properly defined C4I COE is suitable for a service specific system (e.g., Navy JMCIS or Army ABCS) or a joint C4I system (e.g., GCCS). Also, because the architecture is principles-driven, the DII COE is extensible to larger problem domains by expanding the selected set of software components. As discussed in Appendix G, the COE is open and can be applied to non-Unix platforms such as the PC.

Subsection 2.1 discusses fundamental COE concepts and also describes what is meant by COE compliance. As with any standard, compliance is required to avoid conflicts which prevent interoperability. Take careful note in reading subsection 2.1 that the discussion is relevant to any COE-based system since the principles are much broader than a single system such as GCCS or GCSS. Subsections 2.2 through 2.4 elaborate on software and hardware configurations selected for support by DISA, and how the software is structured at a top level to limit site operators to only those functions they are authorized to access. The

final subsection presents an overview of how software is configured at an operational site through use of segments, variants, and profiles.

2.1 Fundamental COE Concepts

In COE-based systems, all software - except the operating system and basic windowing software - is packaged in self-contained units called *segments*. This is true for COE infrastructure software and for mission application software as well. Segments are the most basic building blocks from which a COE-based system can be built. Segments are defined in terms of the functionality they provide, not in terms of "modules," and may in fact consist of one or more CSCIs (Computer Software Configuration Items). The reason for defining segments in this way is that it is a more natural way of expressing and communicating what software features are to be included in, or excluded from, the system than by individual process or file name. For example, it is more natural to think of a system as containing a message processing segment than CSCIs called Icm, Ocm, and Pcm. Those segments which are part of the COE are known as *COE component segments*, or more precisely, as segments which further have the attribute of being contained within the COE. The principles which govern how segments are loaded, removed, or interact with one another are the same for all segments, but COE component segments are treated more strictly because they are the foundation on which the entire system rests.

It is important to note and understand that just because a segment is part of the COE, it is not necessarily always present or required. Considerable flexibility is offered to customize the environment so that only the segments required to meet a specific mission application need are present at runtime. This approach allows minimization of hardware resources required to support a COE-based system. To illustrate the point, consider an example. The COE contains a service for displaying maps. However, some C4I operators in command centers only need to read and review message traffic and do not need, or want, to view a tactical display. Logistics operators, using GCSS, do not need to see the tactical picture at all and may only desire to see a map when planning transportation routes. For such operators, it is not necessary at runtime to have the extra memory and performance overhead of the segments which generate cartographic displays.

It is also important to distinguish between interoperability and integration. In the context of this document, *interoperability* refers to the ability for two systems to exchange data:

- with no loss of precision or other attributes,
- in an unambiguous manner,
- in a format understood by both systems, and
- in such a way that interpretation of the data is precisely the same.

Two systems which are interoperable can exchange data and will produce exactly the same "answer" in the presence of identical data. It is sometimes

useful to describe segments as interoperable, but generally the concern is over system level interoperability. It is the claim of the COE that interoperability can only be achieved when systems use exactly the same software module to perform identical functions. Implementation of agreed upon paper standards is not sufficient by itself.

Integration, within the context of this document, refers to combining segments, not systems. Segment integration refers to the process of ensuring that segments:

- work correctly within the COE runtime environment,
- do not adversely impact one another,
- conform to the standards described in this document,
- have been validated by the COE tools, and
- can be installed on top of the COE by the COE installation tools.

Integration does not imply interoperability; it only provides a level of assurance that the system will work as designed. Integration of a segment with the COE is the responsibility of the segment developer. Integration of the system as a whole and interoperability testing are performed by government integrators.

2.1.1 COE Structure

Figure 2-1 is a simplified diagram which illustrates the relationship between the COE, component segments, and mission application segments. As can be seen, the COE encompasses APIs, GOTS and COTS software, the operating system, windowing software, and standards (*Style Guide*, *I&RTS*, etc.). Physical databases (MIDS IDB, JOPES database, etc.) are not considered part of the COE although the software, such as the RDBMS, which accesses and manages the data is. Figure 2-1 is a generic diagram intended only to show relationships. The labeled boxes show services useful for the C4I and logistics domains, but the principles are the same and the diagram is still valid if the boxes are replaced with examples from another problem domain.

To use a hardware analogy, the COE is a collection of building blocks which form a software "backplane." Segments "plug" into the COE just as circuit cards plug into a hardware backplane. The blocks containing the operating system and windowing environment are akin to a power supply because they contain the software which "powers" the rest of the system. The segments labeled as COE component segments are equivalent to already built boards such as CPU or memory cards. Some of them are required (e.g., CPU) while others are optional (e.g., specialized comms interface card) depending upon how the system being built will be used. The blocks in Figure 2-1 labeled as mission application areas are composed of one or more mission application segments. These segments are

equivalent to adding custom circuit cards to the backplane to make the system suitable for one purpose or another.

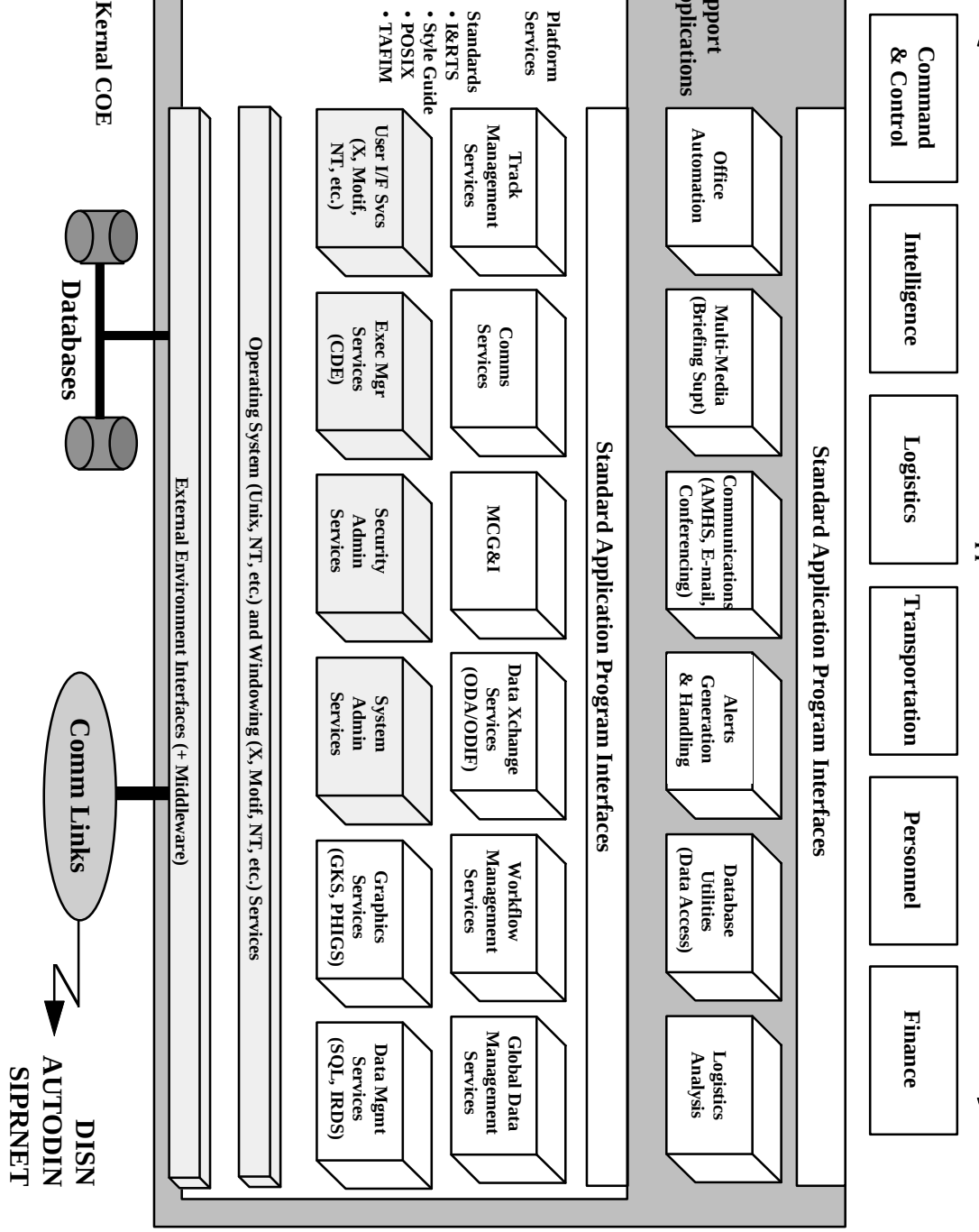

Figure 2-1: DII COE

The API layer shown in Figure 2-1 defines how other segments may connect to the backplane and utilize the "power supply" or other circuit cards. This is analogous to a hardware schematic diagram that indicates how to build a circuit card that will properly plug into the backplane. The figure also implies that APIs are the only avenue for accessing services provided by the COE. This is true for all COE software, including COTS software. However, the COE does not create an additional layer on top of the COTS software. These components may be accessed directly using vendor supplied APIs for these commercial products as long as such usage does not circumvent the intended COE architecture. For example, the COE includes a POSIX-compliant operating system. Some vendors provide non-POSIX compliant extensions to the operating system services. Use of such extensions, even though they are readily available through vendor supplied APIs, is not allowed because such usage violates the intended COE architecture.

This hardware analogy can be extended to implementation of databases within the COE, but with some significant distinctions. Within this conceptual model, the DBMS functions as the COE's disk controller and disk drives. The applications' databases, which are not part of the COE, can be equated to directories or partitions on the drives accessed through the DBMS "disk controller." Data objects belonging to each database then can be considered as files within those "directories."

This analogy is critical to understanding the modularity limitations for databases within the COE. One can change out most peripherals or circuit cards without any side effects just as one can replace mission applications without losing information. However one cannot put in a larger disk drive, or change from one type of controller to another, without losing the data on the disk. While upgrading mission applications is like swapping circuit cards, upgrading databases is like rebuilding a disk or directory structure. Instead of replacing a component, one must save and then restore the files on the disk. Further, each site's database must be presumed to be unique in the same way that disks in different systems have different files.

COE databases are divided among segments as are mission applications, but with a different focus. Mission applications are segmented based on their functionality. Databases are segmented by their content or by the subject area of the mission applications they support. Thus mission applications are functional modules; databases are informational modules.

The precise configuration of COTS products used in the COE is placed under strict configuration control. This is necessary because configurable items such as the amount of shared memory or swap space must be known and carefully controlled in order for other components in the COE to operate properly. For this reason, COTS products are assigned a version number in addition to the vendor

supplied version number so as to be able to track and manage configuration changes.

A fundamental principle throughout the COE is that segments are not allowed to directly modify any resource "owned" by another segment. This includes files, directories, modifications to the operating system, and modification to windowing environment resources. Instead, the COE provides tools through which a segment can request extensions to the base environment. The importance of this principle can not be overemphasized because environmental interactions between software components are a primary reason for difficulties at integration time. By providing software tools which arbitrate requests to extend the environment, integration can be largely automated or at least potential problem areas can be automatically identified.

For example, the COE predefines a set of ports in the Unix `/etc/services` file. Some segments may need to add their own port definitions, but this will create conflicts if the port definitions are the same as those defined by the COE or another segment. To identify and prevent such conflicts, segments issue a request to the COE (see Chapter 5 for how this is done) to add their port definitions. This process is called "environment extension" because a segment is modifying the predefined environment by extension, not through replacement or deletion.

COE component segments shown in Figure 2-1 are typically designed to be servers. However, note that in practice such segments will often operate in both a client and server mode. For example, a track management segment is a server for clients that need to retrieve the current latitude/longitude location of a platform. But the track manager itself is a client to a communications server which initially receives track related reports from sensors or other sources. Refer to the *Architectural Design Document for the Global Command and Control System (GCCS) Common Operating Environment (COE)* document for more detailed discussion of how the COE component segments are designed and interact. For purposes of the present discussion, it is sufficient to view COE segments as servers which are accessible through APIs.

2.1.1.1 COE Installation

The COE will normally make available a large number of segments, not all of which are required for every application. The *kernel COE* is the minimal set of software required on every workstation regardless of how the workstation will be used. The kernel COE components are the lightly shaded boxes in Figure 2-1 and include the Operating System and Windowing Services, and External Environment Interfaces. A kernel COE will always contain the operating system and windowing environment, but it will normally include five other features:

1. a basic System Administration function,
2. a basic Security Administration function,
3. an Executive Manager function (e.g., a desktop GUI),
4. a template for creating privileged operator login accounts, and
5. a template for creating non-privileged operator login accounts.

The System Administration segment is required because it contains the software necessary to load all other segments. The Security Administration segment is required because it enforces the system security policy. The Executive Manager is required because it is the interface through which an operator issues commands to the system. The Executive Manager is an icon and menu driven desktop interface, not a command line interface. The templates included in the kernel COE describe the basic runtime environment context that an operator inherits when he logs in (which processes to run in the background, which environment variables are defined, etc.). The kernel COE assures that every workstation in the system operates and behaves in a consistent manner, and that every workstation begins with the same environment.

From an installation sequence perspective, it is necessary to define a subset of the kernel COE called the *bootstrap COE*. The reason is that segments are installed through a special COE program, the segment installation tool. The segment installation tool is accessed as a System Administration function. However, the segment installation tool itself must be installed before it can be used to install segments. Moreover, as noted above, certain COTS software is not in segment format. How then is the segment installation tool installed, and at least a minimum operating system, to permit loading the kernel COE? This is typically done by loading the operating system and windowing environment as directed by the hardware vendor, then the COE segment installation software is loaded. Once the COE is thus "bootstrapped," it is possible to load the remaining components of the kernel COE and any additional segments.

Figure 2-2 is a more detailed notional depiction of the process. (It should not be interpreted too literally since vendor specific loading instructions may require slight alterations in the loading sequence shown.) First, the operating system, windowing environment, and any necessary patches are loaded as per vendor instructions. Then, the COE Security and System Administration software are copied onto disk with the equivalent of a Unix tar command. The segment installation tool is copied onto disk as part of installing the System Administration software and installation of the System Administration software is done in such a way as to also create a System Administrator operator account. This completes installation of the bootstrap COE. Next, the operator logs in as a system administrator, invokes the segment installation tool and selects the remaining kernel COE segments for installation. Finally, any remaining mission specific segments are selected and loaded.

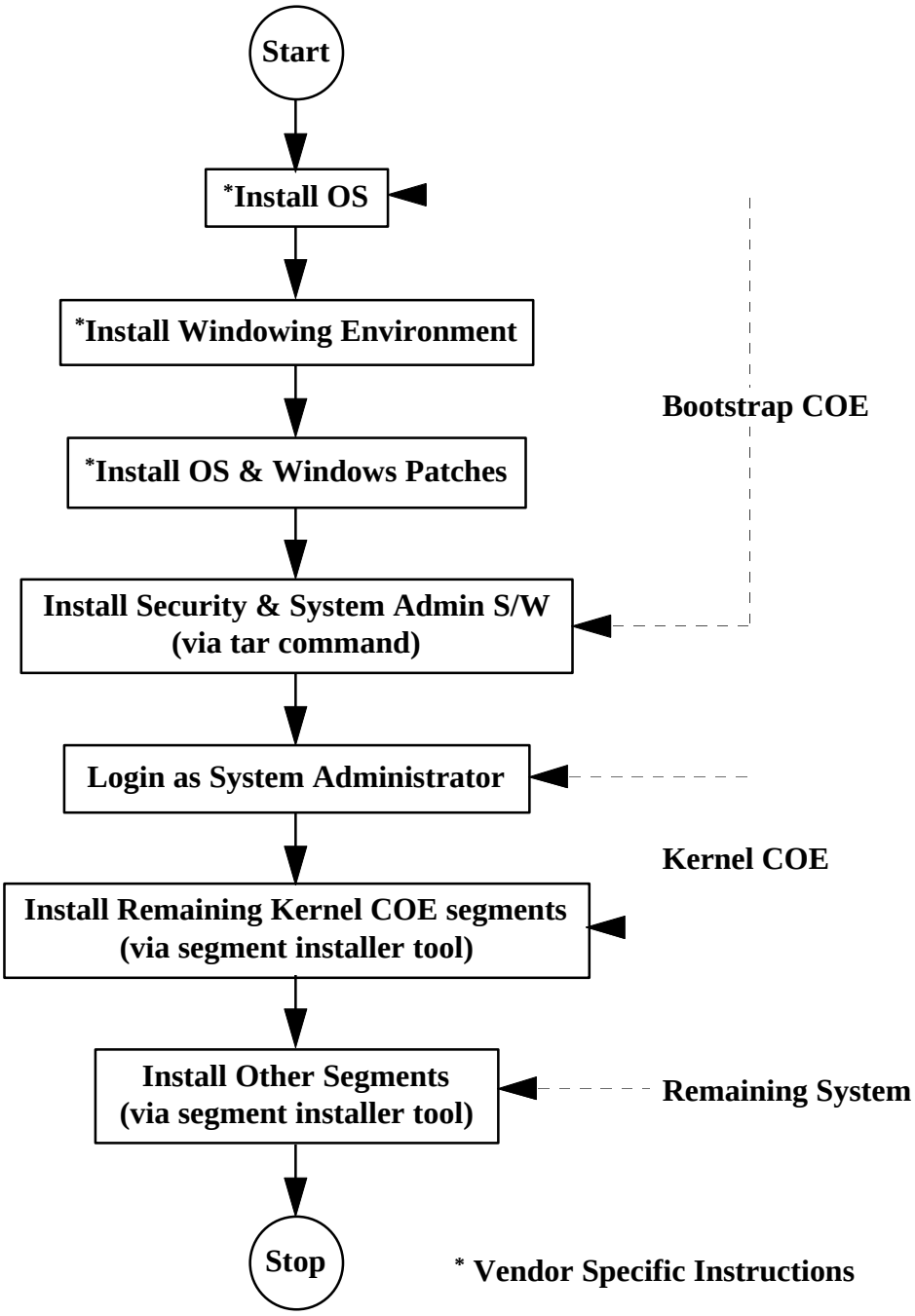

Figure 2-2: Installing the Bootstrap and Kernel COE

This installation approach has several advantages. It greatly simplifies the installation process by handling all vendor unique issues first (e.g., loading the operating system and patches). It guarantees a standard, known starting configuration for all platforms regardless of how they will be used. It allows all remaining segments to be loaded in a standard way regardless of the hardware platform or mission application, thus simplifying system administration. Through the COE, it allows segments to extend the base environment as required as they are loaded.

2.1.1.2 COE "Plug and Play"

The DII COE is structured as a "plug and play" architecture. The key to the "plug and play" design is conformance to the COE through the rules detailed in this document, and through using only the published APIs for accessing COE services. There is considerable danger in using unpublished, "private" APIs, or APIs from legacy systems, because there is no guarantee that interfaces used in this fashion will remain the same or even exist in subsequent releases. This is no different than using vendor supplied COTS products, and the risks are the same.

Discussion of the COE as a "plug and play" architecture is not intended to trivialize the effort which may be required to develop and integrate a segment into the COE. Migration of existing legacy systems to the COE is conceptually straightforward but may require considerable effort due to the requirement to switch to a different set of building blocks. That is, the effort may not be so much in adjusting to a new architectural concept but in adjusting code to use a different set of APIs. The "plug and play" paradigm is a good conceptual model because it clearly conveys the goal and conveys the simplicity that the majority of segment developers will encounter.

2.1.2 COE Principles

Selection of the specific software modules which comprise the COE determines which problem domain(s) can be addressed by the particular COE instantiation. But selection of COE components is not arbitrary; it is driven by a number of important architectural and programmatic principles. First, there is a determination of what functions the COE is required to perform, then there is a set of criteria for selecting software components which perform the required functions. A function is part of the COE if it meets one or more of the following criteria:

1. *The function is part of the minimum software required to establish an operating environment context.* This is normally provided by COTS products and includes Unix, X Windows, Motif, security software, and networking software.

2. *The function is required to establish basic data flow through the system.* To be useful, a system must have a means for communicating with the external world. To be efficient, consistent, and robust, a system must also have standard techniques for managing data flow internal to the system. The COE contains these types of functions. For example, a COE appropriate for a command information system must include communications interfaces, message processing, track management, correlation, tactical display management, database management software, alerts management, and cartographic display functions.

3. *The function is required to ensure interoperability.* Standards alone can not guarantee interoperability, but using common software for common functions comes much closer. As an example from the GCCS problem domain, an OTH-GOLD message parser is part of the COE because interoperability can not be achieved if two different message parsers implement a different set of assumptions about the OTH-GOLD message specification, or use a different specification revision.

4. *The function is of such general utility that if rewritten it constitutes appreciable duplicative effort.* This includes printer services, an alerts service for disseminating alerts (for a command information system), and a desktop environment for launching operator initiated processes.

Subsection 2.2 details the functions currently defined to be in the DII COE. It is presently being expanded to include logistics and financial support functions. The first three criteria listed above are technical in nature because they dictate from an architectural perspective what software *must* be contained in the COE for a given problem domain. The fourth criteria, however, is more programmatic in nature because it is often a tradeoff between the cost of modifying a legacy system to remove duplication versus the cost of maintaining duplicative code, the cost of potentially requiring additional hardware resources because of duplication, and the cost of operator training when there are different ways to accomplish the same action. COE compliance requires that there be no duplication of functions in the first three criteria but some flexibility is possible for the fourth.

There are two frequently voiced perceptions about the COE services:

- a perception that if a module becomes part of the COE, it can not be easily changed or customized, and

- a perception that the larger the COE is, the more inflexible and the poorer the performance of the resulting system.

The first perception stated is partially true and is so by design. COE components must be carefully configuration managed, and changed only in a controlled way in response to formally reported problems. Stability of the COE is crucial to the system, so modifications are done carefully, deliberately, and at a slower pace than changes in non-COE routines. But just because changes are controlled does not mean that the COE routines can not be customized. Ongoing work in the COE is to devise and refine techniques to "open up the architecture" to allow applications to customize COE components in ways that do not violate COE principles and do not adversely impact other developers using COE services.

The second perception stated is a misunderstanding of the COE architecture and concept. Unlike many systems, the COE is not designed as a single monolithic process, but is instead designed as a collection of relatively small processes. While a small number of these are loaded into memory as background processes, most are loaded into memory on demand in response to operator actions (e.g., edit a file, display a parts inventory) and only for the amount of time required for them to perform their task. This approach offers considerable flexibility because it limits the number of background processes required. With the exception of adding new background processes, performance is not adversely impacted by adding new segments. The price paid is a small amount of overhead required to load functions on demand, but this is generally negligible because the overhead is small and is usually in response to an operator request to bring up a display that must respond only at human speeds.

A COE component segment can be omitted from the installed COE if:

- *The functionality provided by the segment is not required by any mission application.* For example, the COE provides a V(6) interface. Sites which do not interface to a V(6) communications device need not include this software. However, in many cases there is no real advantage to deleting a COE component segment because it will not be activated unless required and the amount of disk space taken up is small. Eliminating the function will likely create more configuration management problems than leaving the function in the system.

- *The functionality provided by the segment is not required by any remaining COE component segments.* Selection of certain functions within the COE automatically dictates the inclusion of segments on which the functions depend. This is not the same as saying that the COE is not modular. To the contrary, it is an observation that inclusion of a higher level function requires inclusion of all lower level routines used to build the function. This is a direct consequence of modularity, not a contradiction.

- *The functionality provided by the COE segment is not being omitted because the functionality will be duplicated by other segments.* A common pitfall to avoid is omitting a COE component because its functionality is available through some other means. The problem with this approach is that a common "look and feel" and consistent operation is no longer preserved between applications, and interoperability may be at risk.

Omission of COE component segments which are not used is normally handled automatically by the COE installation software.

2.1.3 COE Compliance

The degree to which "plug and play" is possible is dependent upon the degree to which segments are COE compliant. Appendix B contains a detailed checklist for areas where compliance is mandatory, and a checklist for areas where compliance is ultimately required but for which there is room for a migration strategy to achieve full compliance. The COE provides a suite of tools, described in Appendix C, which validate COE conformance.

By its very nature, an exhaustive list of "do's and don'ts" is not possible. COE compliance must be guided by overarching principles with checklists and tools to aid in detecting as many problem areas as possible. Full COE compliance embodies the following principles:

1. All segments shall comply with the guidelines, specifications, and standards defined in this document and related documents such as the *Style Guide.*

2. All software and data shall be structured in segment format. By definition, COTS components of the bootstrap COE are exempted from this requirement. Segment format is described more fully in Chapter 5.

3. All segments shall be registered and submitted to the on-line library. The registration process is described in Appendix E while submission of segments to the on-line library is described in Chapter 7.

4. All segments shall be validated with the `VerifySeg` tool prior to submission, and shall successfully pass the `VerifySeg` tool with no errors. An annotated listing of the `VerifySeg` tool output shall be submitted with each segment release.

5. All segments shall be loaded and tested in the COE environment prior to submission. Segment developers are responsible for testing their

segment within the full COE, but there is no requirement to include mission application segments for which there is no dependency.

6. All segments shall fully specify dependencies and required resources through the appropriate segment descriptors defined in Chapter 5.

7. All segments shall be designed to be removable, and tested to confirm that they can be successfully removed from the system. Some segments, especially COE components, are designed to be "permanent" but even these must be removable when a later segment release supersedes the current one.

8. All segments shall access COE components only through the published APIs, and segments shall not duplicate functionality contained within the COE. There is no requirement to integrate to COE functionality which is not required by the segment, but note that use of some segments may have an implied dependency on other segments.

9. No segment shall modify the environment or any files it does not own except through environment extension files or through use of the installation tools provided by the COE.

COE compliance can also be addressed by the degree to which integration with COE component segments and the runtime environment has been achieved, from "peaceful coexistence" to "fully integrated." However, the degree of software integration achieved is only one important way of measuring COE compliance. Equally important are measures of GUI consistency, architectural compatibility, and software quality. The DII COE defines four areas of compliance, shown in Figure 2-3, called compliance *categories.* Within a specific category, a segment is assigned a numerical value, called the compliance *level*, which is a measure of the degree to which a segment is compliant within the category. The DII COE takes this approach because it is especially useful to have compliance categories and levels when defining a migration strategy for legacy systems.

Runtime Environment	Style Guide	Architectural Compatibility	Software Quality
0 → j	0 → k	0 → n	0 → m

Figure 2-3: COE Compliance Categories and Levels

The four DII COE compliance categories are:

Category 1: Runtime Environment. This category measures how well the proposed software fits within the COE executing environment, and the degree to which the software reuses COE components. It is an assessment of whether or not the software will "run" when loaded on a COE platform, and whether or not it will interfere with other segments.

Category 2: Style Guide. This category measures how well the proposed software operates from a "look and feel" perspective. It is an assessment of how consistent the overall system will appear to the end user. It is important that the resulting COE-based system appear seamless and consistent to minimize training and maintenance costs.

Category 3: Architectural Compatibility. This category measures how well the proposed software fits within the COE architecture (client/server architecture, DCE infrastructure, CDE desktop, etc.). It is an assessment of the software's potential longevity as the COE evolves. It does *not* imply that all software must be client/server and RPC (Remote Procedure Call) based. It simply means that a reasonable design choice has been made given that the COE is client/server based and is built on top of a DCE (Distributed Computing Environment) infrastructure.

Category 4: Software Quality. This category measures traditional software metrics (lines of code, McCabe complexity metric, etc.). It is an assessment of program risk and software maturity.

To declare that a segment is COE-compliant requires qualification with a category name and a compliance level, and is meaningless otherwise. These four categories attempt to quantitatively answer the following questions about a proposed addition to the system:

- Can the proposed software be added to the system?
- Is the proposed software user-friendly?
- Is the proposed software architecturally sound and in line with where the COE is going?
- What is the program risk?

The remainder of this document will address only Category 1 - Runtime Environment.

The COE defines eight progressively deeper levels of integration for the Runtime Environment Category. Note that levels 1-3 are "interfacing" with the COE, not true integration. Integration begins at level 4.

Level 1: Standards Compliance Level. A superficial level in which the proposed capabilities share only a common set of COTS standards. Sharing of data is undisciplined and minimal software reuse exists beyond the COTS. Level 1 may allow simultaneous execution of the two systems.

Level 2: Network Compliance Level. Two capabilities coexist on the same LAN but on different CPUs. Limited data sharing is possible. If common user interface standards are used, applications on the LAN may have a common appearance to the user.

Level 3: Workstation Compliance Level. Environmental conflicts have been resolved so that two applications may reside on the same LAN, share data, and coexist on the same workstation as COE-based software. The kernel COE, or its equivalent, must reside on the workstation. Segmenting may not have been performed, but. some COE components may be reused. Applications do not use the COE services and are not necessarily interoperable.

Level 4: Bootstrap Compliance Level. All applications are in segment format and share the bootstrap COE. Segment formatting allows automatic checking for certain types of application conflicts. Use of COE services is not achieved and users may require separate login accounts to switch between applications.

Level 5: Minimal COE Compliance Level. All segments share the same kernel COE, and functionality is available via the Executive Manager. Boot, background, and local processes are specified through the appropriate segment descriptor files. Segments are registered and available through the on-line library. Applications appear integrated to the user, but there may be duplication of functionality and interoperability is not guaranteed. Segments may be successfully installed and removed through the COE installation tools.

Level 6: Intermediate COE Compliance Level. Segments utilize existing account groups, and reuse one or more COE component segments. Minor documented differences may exist between the *Style Guide* and the segment's GUI implementation.

Level 7: Interoperable Compliance Level. Segments reuse COE component segments to ensure interoperability. These include COE provided comms interfaces, message parsers, database tables, track data elements, and logistics services. All access is through published APIs

with documented use of few, if any, private APIs. Segments do not duplicate any functionality contained in COE component segments.

Level 8: Full COE Compliance Level. Proposed new functionality is completely integrated into the system (e.g., makes maximum possible use of COE services) and is available via the Executive Manager. The segment is fully compliant with the *Style Guide* and uses only published public APIs. The segment does not duplicate any functionality contained elsewhere in the system whether as part of the COE or as part of another mission application segment.

Bootstrap Compliance (Level 4) is required before a segment may be submitted to DISA for evaluation as a prototype. Such segments will not be fielded nor accepted into the on-line library. At DISA's discretion, segments which meet the criteria for Minimal COE Compliance (Level 5) may be accepted into the on-line library, and installed at selected sites as prototypes for user evaluation and feedback. Such segments will not be accepted as fieldable products. Acceptance as an official DISA fieldable product requires demonstration of Interoperable Compliance (Level 7) and a migration strategy to Full COE Compliance (Level 8), unless the proposed segment is an interim product that is targeted to be phased out in the near term.

The compliance categories and levels defined here are a natural outcome of developing a reasonable approach to migrating legacy systems into the COE. The first step of Category 1, covered by Levels 1-4, is to ensure that systems do not destructively interfere with each other when located at the same operational site. Level 5 is sometimes called a "federation of systems" in that systems are still maintained as "stovepipes," but they can safely share common hardware platform resources. Levels 6-8 complete the approach by reducing functional duplication, promoting true data sharing, and making the system appear to the user as if it were developed as a single system. The last three levels represent varying degrees of integration from marginally acceptable (Level 6) to a truly integrated system (Level 8).

2.2 COE Component Segments

COE component segments can be categorized in several ways. The original GCCS COE was subdivided into 19 functional areas and was organized largely by technologies employed such as network, database, and MCG&I (Mapping, Charting, Geodesy, and Imaging). Working groups were established for each of the 19 functional areas to consolidate operational requirements from each of the services/agencies, and to evaluate and recommend candidate modules as core components. This taxonomy was initially successful and led to several early successes. However, the large number of working groups defined by this taxonomy quickly became unwieldy and communication within and between working groups became infeasible.

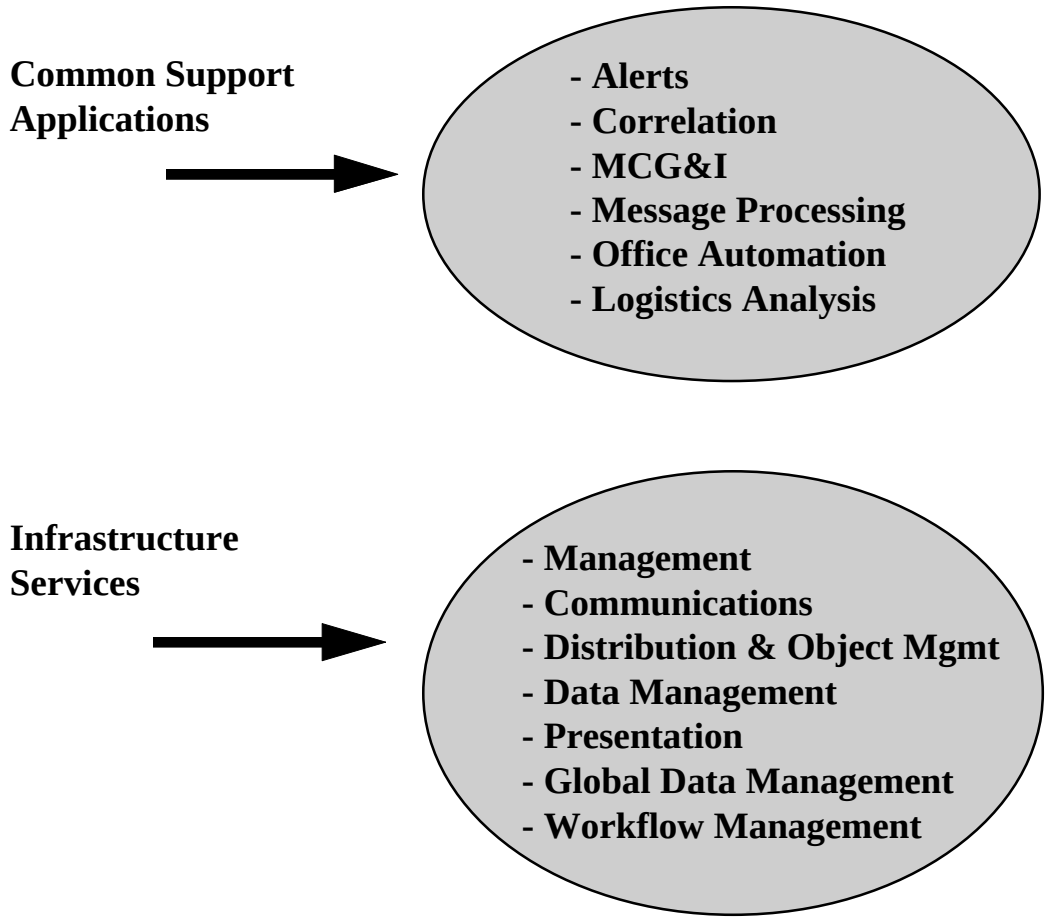

Figure 2-4: DII COE Taxonomy

The present COE taxonomy is detailed in the *Architectural Design Document for the Global Command and Control System (GCCS) Common Operating Environment*

(COE). It approaches the taxonomy from a perspective of providing a collection of Infrastructure Services and a collection of Common Support Applications. These are summarized in Figure 2-4, and extended to include logistics services. While encompassing the same functionality as the original 19 functional areas, this taxonomy approaches the problem from an architectural perspective rather than functional, and greatly reduces the communications burden in and between working groups. Figure 2-4 is presently being extended to formalize logistics support services, and to include services for financial systems (e.g., EC/EDI).

Infrastructure Services provide the architectural framework for managing and distributing the flow of data throughout the system. Management Services include network, system, and security administration. Communications Services provide facilities for receiving data external to the system, and for sending data out of the system. Distribution and Object Management Services provide the infrastructure necessary to achieve true distributed processing in a client/server environment. Data Management Services include relational database management as well as file management in a distributed environment. Presentation Services are responsible for direct interaction with the human whether that be through windows, icons, menus, or multi-media. Workflow and Global Data Management Services are oriented towards managing logistics data (parts inventory, work in process, etc.).

Infrastructure Services originated from the C4I problem domain, but the services provided are largely independent of any particular application. Common Support Applications tend to be much more specific to a particular problem domain. The Alerts Service is responsible for routing and managing alert messages throughout the system whether the alert is an "out of paper" message to a systems administrator or an "incoming missile" alert to a watch operator. The Correlation Service is responsible for maintaining a consistent view of the battle space by correlating information from sensors or other sources that indicate the disposition of platforms of interest. MCG&I Services handle display of Defense Mapping Agency maps or other products, and imagery received from various sources. Message Processing Services handle parsing and distribution of military format messages. Office Automation Services handle word processing, spreadsheet, briefing support, electronic mail, World-Wide Web browsers, and other related functions. Logistics Analysis contains common functions, such as Pert charts, for analyzing and displaying logistics related information.

Selection of software modules which fulfill these COE component responsibilities, and extending them to meet other problem domains, is an on-going task as is the evolutionary nature of the COE. Even though the process is evolutionary, the COE preserves backwards compatibility so that mission applications are not abandoned just because there is an update of the COE. Refer

to the appropriate API, User's Guide, and system release documents for detailed information on the components currently selected for the COE.

2.3 Supported Configurations

The DII COE is an open architecture and as such is not tied to a specific hardware platform. It uses POSIX-compliant operating systems and industry standards such as X Windows and Motif. In actual practice, POSIX compliance and industry standards have not progressed to the point where verification that software works in one hardware/software configuration is a guarantee that it will work in another. COTS vendors do not necessarily provide backwards compatibility with subsequent releases, and in fact much of the effort consumed in porting the COE from one configuration to another is to account for lack of compatibility between vendors, or between vendor releases. Thus, what hardware/software configurations to support is more of a testing and life cycle maintenance issue than it is one of openness or software portability.

The list of supported hardware and software components is growing as the COE and COE-based systems evolve to meet operational requirements. Appendix A lists the current DISA supported COE configurations. This appendix will be updated as required to reflect new hardware/software configurations. Note that not all of the COTS products listed in Appendix A are part of the kernel COE and thus are not required for every workstation. Refer to the DISA Chief Engineer for requirements for other platform or COTS software versions, or for an updated list of supported vendor products.

Precise hardware requirements in terms of memory, disk space, etc. is a function of whether the workstation is a database server or client workstation, and whether the workstation is standalone or on a network with other COE-based workstations. Refer to the DISA Chief Engineer for hardware configuration options.

2.4 Account Groups

In the Unix operating system, users are normally assigned individual login accounts with various configuration files such as `.login` and `.cshrc` that establish a runtime environment context. COTS products such as X Windows and Motif may also have configuration files, such as `.Xdefaults` and `.mwmrc`, that establish a runtime environment context as well. These configuration files must be set up and established for each user of the system. An *account group segment* is a template used within the COE for setting up individual login accounts. Account groups contain template files for defining what COE processes to launch at login time, what functions are to be made available to operators, and preferences such as color selections for window borders. Account groups are described further in Chapter 5 of this document.

Account groups can also be used to perform a first level division of operators according to how they will use the system. This technique is used in the COE to identify at least five distinct account groups:

- ¥ Privileged Operator (e.g., root) Accounts,
- ¥ System Administrator Accounts,
- ¥ Security Administrator Accounts,
- ¥ Database Administrator Accounts, and
- ¥ Non-Privileged Operator Accounts.

Other account groups may exist for specialized system requirements, such as providing a character based interface, but all account groups follow the same rules. Within an account group, subsets of the available functionality can be created. These subsets are called *profiles.* An operator may participate in multiple account groups with multiple profiles, and can switch from one profile to another without the need to log out and log in again. Figure 2-5 shows the hierarchical relationship among account groups, profiles, and individual users.

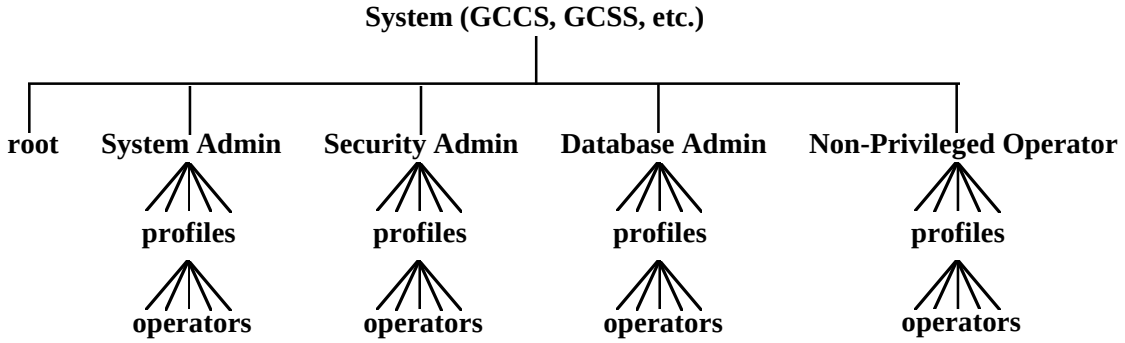

Figure 2-5: Account Groups, Profiles, and Users

2.4.1 Privileged User Accounts

Unix systems provide a "super user" account called root. Its use is normally restricted to knowledgeable systems administrators because serious damage can be done to the system if used improperly. Security requirements also dictate careful control and auditing of actions performed when operating as a privileged user.

The COE design philosophy is to not require the use of a privileged user account for normal operator activities. Certain processes (such as setting the system time) can not be performed without superuser privileges, but such privileges are given to the process, not the user, and only for the period of time necessary to perform the required action. Root level access is *not* provided to the user for such actions.

Normal operation does not require a command-line level access to root. For security reasons, command-line access is expressly prohibited unless prior approval is granted by the DISA Chief Engineer. However, a root account is preserved in the system for use by trusted processes, for unusual system administration tasks or installations, and for abnormal situations where "all else fails."

2.4.2 Security Administrator Accounts

Security in the COE is implemented through a Security Server and through a special trusted security client. This client is the Security Administrator application, which is an interface to the Security Server, that allows a Security Administrator to monitor and manage security. Precise functionality of the Security Service is hardware-dependent because different approaches have been taken by different vendors to provide Unix security.

The Security Service is loaded as part of the bootstrap COE. The Security Administrator application is designed to be made available to only a restricted group of operators. Available functions include the following:

- ¥ Ability to create individual login accounts
- ¥ Ability to create defined operator profiles
- ¥ Ability to customize menus by operator profile
- ¥ Ability to export accounts and profiles to other LAN workstations
- ¥ Ability to review and archive the audit log.

2.4.3 System Administrator Accounts

The System Administrator Account Group is a specialized collection of functions that allow an operator to perform routine maintenance operations. This software is designed to be made available to a restricted group of operators. It is loaded as part of the bootstrap COE because it contains the software required to load segments. Functionality provided includes:

- ¥ Ability to format floppy disks
- ¥ Ability to install and to remove segments
- ¥ Ability to set workstation name and IP address
- ¥ Ability to install and configure printers
- ¥ Ability to create and to restore backup tapes
- ¥ Ability to shutdown and to reboot system
- ¥ Ability to configure DDN host tables
- ¥ Ability to configure and manage the network.

2.4.4 Database Administrator Accounts

The Database Administrator Account Group is to be used by those individuals responsible for performing routine database maintenance activities such as backups, archives, and reloads. The specific capabilities are dependent upon which commercial relational database software is in use and upon tools provided with these commercial products.

Functions included within this account group are:

- ¥ Ability to archive and restore database tables
- ¥ Ability to import and export database entries
- ¥ Ability to checkpoint and journal database transactions.

2.4.5 Operator Accounts

Most operators will not require, nor will site administrators grant access to, capabilities described in the previous subsections. Most system users will be performing mission specific tasks such as creating and disseminating Air Tasking Orders (ATOs), preparing briefing slides, performing ad hoc queries of the database, participating in collaborative planning, etc. The precise features available depend upon which mission application segments have been loaded, and the profile assigned to the operator.

2.5 Site Configuration

Figure 2-6 is a pictorial representation of several COE terms that relate to how software is configured at an operational site. The *superset* is the total collection of all software, data, and COTS products in the COE and COE-based systems. It is neither desirable nor feasible to install all software and all data on every workstation. Instead, only a subset of the segments are installed on any particular workstation. The kernel COE is required to be on each workstation, but additional segments are dependent upon how the workstation will be used. The exact configuration of segments installed is called a *variant*. A variant is simply a collection of segments that are grouped together for installation convenience. For example, it is more convenient for an operator to indicate that a workstation is to act as a database server (a variant), or used as an intelligence analyst workstation (another variant) than to manually select all of the segments that need to be installed. The COE is designed so that a site may install predefined variants, or can customize the installation to suite site specific requirements.

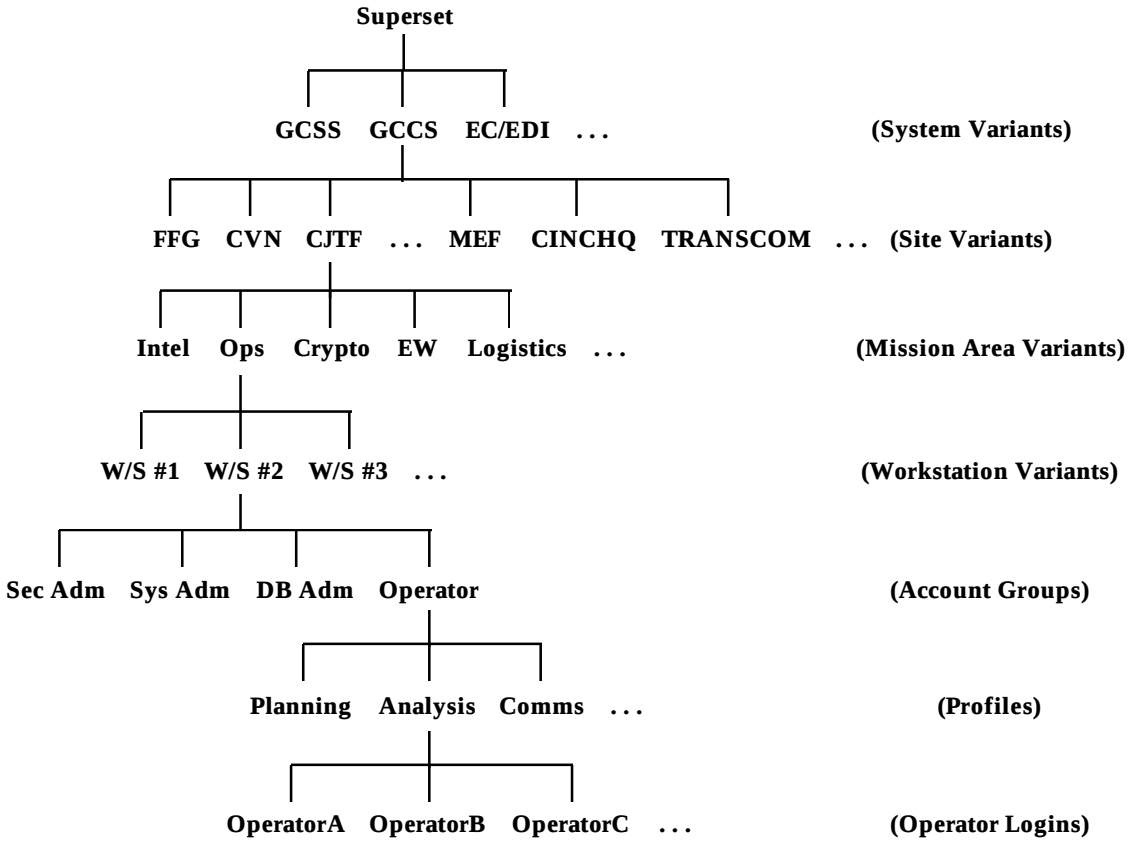

Figure 2-6: Configuration Terminology

Figure 2-6 shows four types of variants. A *system variant* is a collection of segments from the superset which are required to provide functions for a very specific problem domain (e.g., GCCS for C4I, GCCS for logistics). A *site variant* is a subset of segments from the system variant which are required to meet site specific mission objectives (e.g., JTF, USACOM, USPACFLT, TRANSCOM). Sites will often group workstations together by mission area into physical spaces, rooms, or other logical arrangements. In the example shown for a JTF, GCCS will be installed in Intelligence, Operations, Environmental spaces, etc. This subdivision of segments forms a *mission area variant*. Within a space, GCCS will be installed on one or more network workstations. These are called *workstation variants.*

As can be seen from Figure 2-6, a system variant is a list of site variants, a site variant is a list of mission area variants, a mission area variant is a list of workstation variants, and a workstation variant is a list of segments. Unless the distinction is important, a variant will normally mean just a list of segments with the implied understanding that it may actually be a list of variants which ultimately decompose into a list of segments.

The same media can be used to load any workstation regardless of which site or in which space the workstation is located; however, during the installation process, only that portion of the variant required for a particular workstation is actually loaded. The kernel COE is a required member of every variant.

Once loaded onto a workstation, the system functionality provided by the workstation variant is further subdivided into account groups. However, operators will not normally require access to the total functionality within a particular account group due to security considerations or Standard Operating Procedures (SOPs) in effect at the site. That is, each account group will usually be subdivided in to one or more operator profiles. Operators that have the same profile may have different preference files or security access privileges that are determined by the individual operator's login.

For example, in Figure 2-6 the CJTF site variant may have mission area variants defined for Ops, Intel, Crypto, etc., spaces. Within a space, accounts groups may be installed for security, system administration, database, and watch officer operations. Profiles might be designated to group operators into Planning, Analysis, or Comms. Within a specific profile, individual password-protected login accounts are assigned to actual system users.

There are several advantages to this approach:

- From a configuration management and security perspective, only one set of distribution media is required to be controlled.

- From an installation perspective, the site installer only has one set of distribution media to worry about regardless of workstation use or hardware type. (The COE tools allow segments for multiple platforms to exist on the same physical distribution media. At installation time, the software determines the platform type and then makes available for selection only those segments which can execute on the platform.)

- From a system design perspective, the ability to create variants allows the flexibility of loading and executing only that software which is required to support a particular mission requirement.